AD-A071 105 SYSTEM DEVELOPMENT CORP SANTA MONICA CALIF F/G 5/2
A DEDUCTIVE CAPABILITY FOR DATA MANAGEMENT,(U)
1976 C KELLOGG, P KLAHR, L TRAVIS N00014-76-C-0885
UNCLASSIFIED NL

1 OF 1
AD
A071 105

END
DATE
FILMED
8-79
DDC

AD A071105

Contract N00014-76-C-0885

Systems for Large Data Bases, P.C. Lockemann and E.J. Neuhold, (eds.)
North-Holland Publishing Company, 1976

LEVEL

A DEDUCTIVE CAPABILITY FOR DATA MANAGEMENT

Charles Kellogg, System Development Corporation, Santa Monica, Calif.
Philip Klahr, System Development Corporation, Santa Monica, Calif.
Larry Travis, University of Wisconsin, Madison, Wisconsin

This paper examines some of the problems and issues involved in designing a practical deductive inference processor to augment a data management system, as well as some of the benefits that can be expected from such an augmentation. A deductive processor design is presented that incorporates new techniques for selecting, from large collections of mostly irrelevant general assertions and specific facts, the small number needed for deriving an answer to a particular query.

INTRODUCTION

In this paper we discuss some of the issues involved in adding a deductive capability to a data management system, and we describe a specific approach towards achieving this objective. Figure 1 illustrates the major components of our deductive data management system:

- A language processor that translates user input into a formal intermediate symbolism.
- A data management system* that retrieves specific facts (n-tuples of data values) from a data base as required.
- A deductive processor that uses general assertions (i.e., premises representing general rule-based knowledge about a data-base domain) to derive implicit information from collections of explicit data values.
- A control module that facilitates communication between the several components of the system and directs interaction between the deductive processor and the data management system during on-line question answering.

Listed below are some of the benefits to be expected from adding a deductive processor to a data management system:

- A deductive processor permits the extraction of information that is not explicitly stored but that can be inferred by combining specific facts in the data base with rule-based knowledge encoded in general assertions. This augmentation of the information-retrieval function can be especially important for very large data-base domains.

DATE 1976

*In our prototype we use a relational data management system (see Codd (1970), Date (1975)). The research described in this paper is an outgrowth and extension of our earlier research on natural-language data management (see Kellogg et al. (1971), Travis et al. (1973)).

DDC FILE COPY

DDC
RECEIVED
JUL 13 1979
A

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DDC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

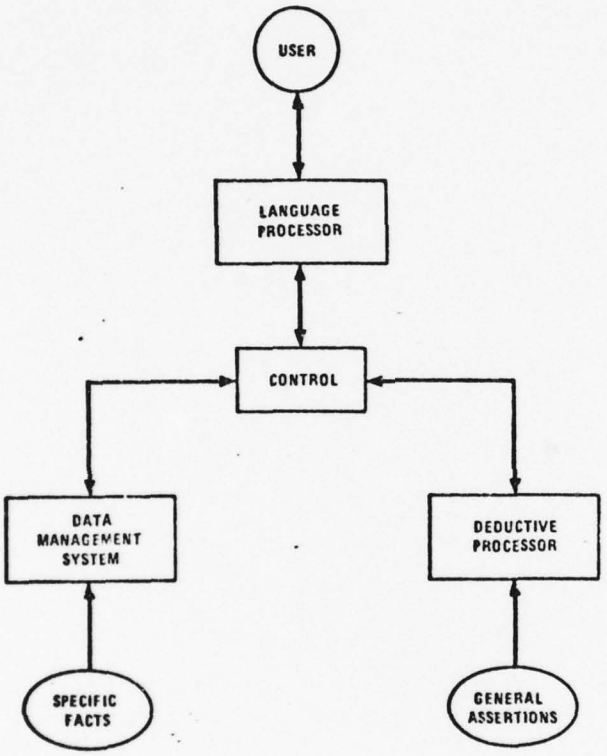

Figure 1. Deductively Augmented Data Management.

Accession For
NTIS GRA&I
DDC TAB
Unannounced
Justification
Letter on file
By
Distribution/
Availability Codes
Dist. | Avail and/or special
A | 23

- A deductive processor allows a data management system's language to be extended and adapted to the needs of particular users. Thus a user's language can be uncoupled from the particular terms and categories used in organizing a data file. This is essential if users are to be able to use a system without having a thorough knowledge of its file structure. For example, a user should be able to ask whether the maternal grandfather of John Kennedy was richer than his paternal grandfather without knowing the respective men's names or that the file is structured in terms of net worth. Relatively powerful inferential mechanisms are necessary to enable full use of such descriptive references.

- A deductive processor not only generates answers to specific queries but also supplies evidence (lines of reasoning) for or against these answers. In some cases, the system may supply one argument leading to "yes" and another leading to "no" (indicating inconsistent information). In the real world of unreliable reports and uncertain facts, this kind of response will in many cases be much more useful to a user than simple categorical answers.

- While deduction is itself a precise and strict process, deductive arguments can use premises of differing degrees of plausibility. Since the plausibility of a conclusion (answer) is a function of the plausibility of the premises from which it is derived, deduction provides a basis for using "soft" information in a computer-based system. The important thing is that the system be able to show the user the evidence for a conclusion as well as the conclusion itself. Deduction can also be used to generate multiple distinct arguments for a conclusion, thereby supplying additional evidence for the plausibility of its answer.

- A deductive processor can, under certain circumstances, supply conditional answers when specific direct answers are not possible; e.g., "Is Joe Smith eligible for a pension?--Yes, if he has thirty years of continuous service." In this case the deductive processor identifies a specific fact about Joe Smith that it needs to complete an argument but that it cannot find in the files to which it has access.

- A deductive processor can answer "what-if" and other kinds of high-level queries that are difficult if not impossible for present-day data management systems.

Each of these capabilities is currently demonstrable within our prototype deductive data management system. Our primary concern in this paper is to outline our deductive system and to give examples emphasizing the derivation of implicit information.

Research on mechanizing deduction has been conducted primarily in the areas of question-answering and theorem proving within the broader area of artificial intelligence. Early question-answering systems such as SIR (Raphael (1964)), PROTOSYNTHEX (Schwarcz et al. (1970)), and CONVERSE (Kellogg (1968, 1971)) relied primarily on set-inclusion logic for their deductive capability. Elliott (1965) developed structure-specific procedures (such as one for transitivity and symmetry) for deriving new information from a file of specific facts. The inferential capabilities in these systems were limited and used for special purposes.

With the development of "resolution" (Robinson (1965)), more sophisticated theorem-proving techniques were incorporated into question-answering systems, most notably in Green (1969) and in Minker et al. (1973). General statements formulated in a first-order predicate-calculus symbolism could now be used to derive new information. Deductive power increased considerably, but at the expense of increased

search space. This led to a host of resolution strategies (Chang and Lee (1973)). Some recent approaches to deduction have offered alternatives to resolution; these include procedure-oriented deductive systems, exemplified by PLANNER (Hewitt (1971)), and natural-deduction systems, exemplified by Bledsoe (1974) and Nevins (1974).

Our primary concern has been to design a deductive processor that will support practical data management in realistic environments involving large files of general and specific information.* We have concentrated on the problem of selecting from such large files the few premises and facts that are relevant for a particular required deduction. We have adopted some of the deductive techniques used in question-answering systems and modified them to be more suitable for data management; we have also introduced new "planning" techniques for premise selection. These techniques are discussed below.

INFORMATION STRUCTURES TO SUPPORT DEDUCTIVE DATA MANAGEMENT

Figure 2 illustrates the principal files and processors that constitute our deductive system. Note that the deductive processor operates primarily on general assertions in its construction of proofs. The data management system accesses and retrieves specific facts when such facts are needed for proof completion. The four files used by the deductive processor are the general assertion file, the predicate connection graph, the variable substitution file, and the semantic advice file. These files have been segmented for purposes of processing efficiency and data organization.

General Assertion File. The deductive processor has access to a file of general assertions, or premises. These premises are represented in a Skolemized, quantifier-free form, as "primitive conditional" expressions. Primitive conditionals are logical statements whose major connective is the implication sign. On either side of this connective, groupings of literals may be combined conjunctively or disjunctively. Each literal is an atomic formula (i.e., a predicate and its arguments) or a negated atomic formula. The primitive conditional is a canonical form for the first-order predicate calculus. This form facilitates finding chains of deductively linked middle-term predicates, displaying inference plans and evidence chains, and storing information in such a way that the strategic or heuristic implications of the original formulation are not lost in the system, as is often the case with other canonical forms (e.g., the conjunctive normal form used in resolution).

A predicate occurrence is uniquely identified by specifying the premise in which it occurs, the predicate name of which it is an occurrence, its ordinal position in the premise, whether it is on the left or right of the main conditional, whether it is negated or not, and whether it is a member of a conjunctive or disjunctive set. For each predicate occurrence, the above information is represented by a unique compact bit string (a single IBM 370 computer word in our current implementation).

Predicate Connection Graph. The predicate connection graph is abstracted from the information available in the premises. Nodes represent predicate occurrences. Each edge between a pair of nodes represents a possible deductive interaction between the predicate occurrences in the nodes. The predicate connection graph is of key importance in our system. It represents explicitly and compactly a great deal of detailed structural information about general assertions and their possible interconnections. This information is in a form that can be quickly accessed and scanned.

---

*For the related artificial-intelligence problem of efficiently using a very large knowledge base, see McDermott (1975) and Fahlman (1975).

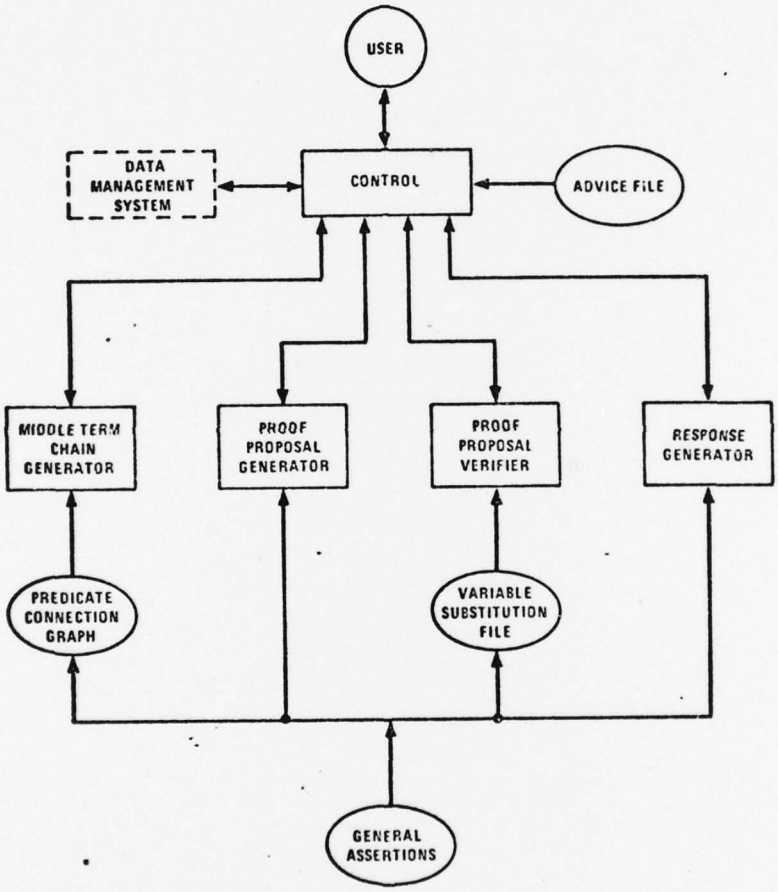

Figure 2. Deductive Data Management System Components.

The predicate connection graph bears some resemblance to the graph proof procedures of Kowalski (1975), Shostak (1976), and Yates et al. (1970). The important distinction between our approach and these procedures is in the way the connection graph is used. As we will see in the next section, the predicate connection graph is used to develop possible proof plans. The graph is an abstraction of information about premises and their deductive interactions and does not, by itself, construct proofs. It is used as a planning tool. For a full discussion of the predicate connection graph, its use in constructing proof plans, and the use of semantic advice in restricting searches through it, see Klahr (1975).

Figure 3 illustrates a small set of premises and their representations in primitive-conditional form. Figure 4 illustrates the predicate connection graph for these premises. The solid lines in Figure 4 are u-arcs (these "unification" arcs represent deductive interactions between different occurrences of the same predicate). These arcs are computed when premises are first entered into the system. Another kind of information in the predicate connection graph is the deductive dependency link, which represents deductive dependency between predicate occurrences within a single premise. Two of the four kinds of dependency links are shown in Figure 4.

Variable Substitution File. Another file, also separated out for purposes of efficiency, is the variable substitution file. This file is also abstracted from information in the premises. It consists of the substitutions for variables that establish the unifications represented by the u-arcs in the predicate connection graph. This file is used only during the verification process, when the substitution lists for all the unifications in a proof are combined and checked for consistency.

Semantic Advice File. Semantic advice can be of considerable aid in deductive searching. It permits the specification and use of deductively significant procedural semantic information specific to a particular domain of discourse. Frequently, advice cannot be formulated directly within the logical symbolism of the general assertions, even when a symbolism as rich as that of primitive conditionals is used. Whenever such advice can be captured, it can very likely be put to good use in simplifying and speeding up the deductive process. In the advice file, semantic advice is formulated and stored as condition-action pairs. (The user may also give problem-specific advice for any particular query.)

DEDUCTIVE PROCESSOR MODULES

Control. The control module provides the primary interface between a user's symbolic input (queries, advice, and data), the several deductive processor modules, and the data management system. For example, Control may locate semantic advice in the advice file relevant to a specific input query. It may then call the Middle-Term Chain Generator and the Proof Proposal Generator to create proof proposals. Control will then invoke the Proof Proposal Verifier and the Data Management System in sequence to verify and complete a proof. Finally, it will call the Response Generator and display the answer and proof to the user.

Middle-Term Chain Generator. The predicate connection graph is used to find chains of middle-term predicates that deductively link the assumption and goal predicates of a query. The basic proof strategies of natural deduction, proof-by-contradiction, and proof-by-cases are automatically incorporated into this predicate chain generator (since the predicate dependency links reflected in the predicate connection graph are of the different kinds needed for all of these strategies; see Klahr (1975)). The chain-generation process may be visualized as one of generating a series of expanding "wave fronts" from each assumption and goal predicate. These wave fronts represent deductively significant possible paths from each predicate. As the two wave fronts expand, intersections are taken to determine when an assumption wave front impinges upon a goal wave front. When this happens, the system has discovered the beginning of a proof plan.

1. Husbands and wives are married to each other.
   $v(H(x_1,x_2), W(x_1,x_2)) \Rightarrow M(x_1,x_2)$
2. Marriage is a symmetric relation.
   $M(x_3,x_4) \Rightarrow M(x_4,x_3)$
3. Spouses of Greeks are Greek.
   $\&(G(x_5), M(x_5,x_6)) \Rightarrow G(x_6)$
4. People living in a place located in Greece are Greek.
   $\&(Loc(x_7,Greece), Liv(x_8,x_7)) \Rightarrow G(x_8)$
5. Spouses live in the same place.
   $\&(M(x_9,x_{10}), Liv(x_9,x_{11})) \Rightarrow Liv(x_{10},x_{11})$

Figure 3. Sample Premise Set in Primitive-Conditional Form.

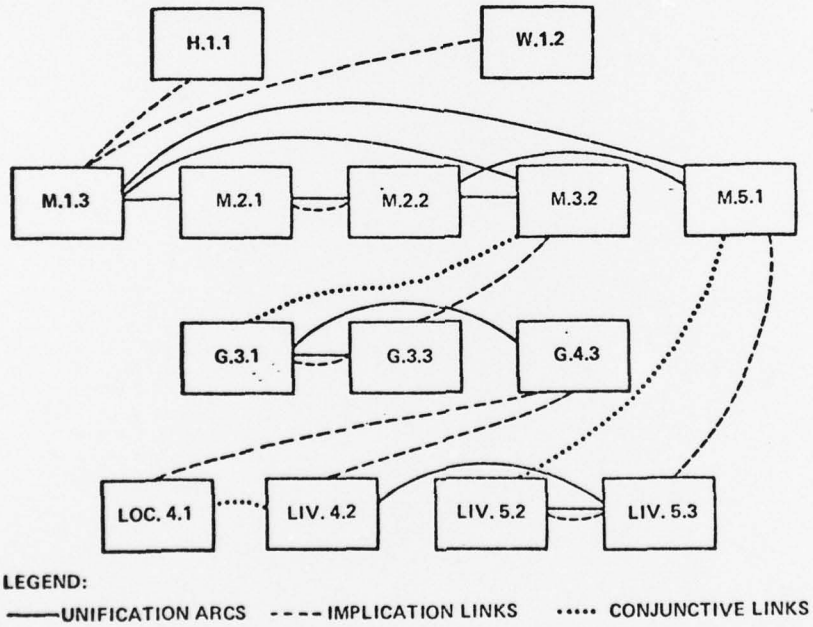

Figure 4. Predicate Connection Graph for the Sample Premise Set. (P.n.m is the occurrence of the predicate P in premise n, position m.)

Proof Proposal Generator. For each middle-term chain produced, the Proof Proposal Generator extracts the premises containing the predicate occurrences in the chain and forms skeletal proof plans. These plans are later sent to the verifier, which constructs full deductive detail for the proofs.

Proof Proposal Verifier. For each proof proposal, the variables and substitutions for them in the proof structure are examined to determine whether there are any blockages (variables taking on conflicting values). If verification is successful, the Control processor examines the proof to see whether there are any remaining subproblems that need support from the file of specific facts. If facts are needed, the data management system is called to search for these facts to complete the proof.

Response Generator. When the system completes a successful verification and instantiation of a proof proposal, it outputs answers and, if desired, the derivations on which they are based. When derivations are not complete, the system may display conditional answers or partial derivations that, in many cases, will provide clues to missing information that the user may be able to acquire from some other source.

A BUSINESS INFORMATION EXAMPLE

As an example of how our system works, let us assume that we have the task of maintaining as complete and accurate a picture as possible of the operation of a large business organization. This will include the need to understand the various factions, real decision-making loci within the organization, and real flows of control and information, as opposed to publicly announced ones or the ones that appear on a formal organization chart as in Figure 5. This chart is for a fictional company that is a large distributing organization with three major line divisions: Chemical, Drug, and Liquor.

We should stress that high-level questions such as "Is the Drug faction or the Liquor faction in supremacy?" have to involve a considerable amount of human interpretation. A user should not expect a computer system, even one capable of sophisticated deduction, to generate categorical answers directly for such questions. What a deductive data management system can do, however, is help to collect and organize evidence for or against a general conclusion or working hypothesis. It is important to note that an inference system may make use of both certain and plausible information in the generation of arguments and in the display of evidence. That is, the evidence may be strong or weak; the human interpreter must judge which.

Let us suppose that a new man, Zembruski, has been appointed executive vice president and head of the Chemical division. We know something about him but this information is spotty and incomplete. The task is to work out deductive connections that might provide evidence toward concluding whether his appointment should be considered a victory for the Drug faction or the Liquor faction. For purposes of simplicity, we will focus on one principal relationship that might bear on this question. This is the notion of informal information flow among individuals. We take this notion to cover the possible exchanges of information between people who are friends, relatives, co-workers, spouses, etc. If we can obtain information that will allow us to deduce various instances of information flow, we might gain evidence, for example, that Zembruski has many more informational contacts with executives in the Liquor division than he does with those in the Drug division.

Suppose that Engler is vice president and head of Liquor. We can ask the question, "Will there be information flow between Engler and Zembruski?" Notice that while this is a yes/no question, we will not be satisfied with just a simple "yes" or "no." We will want to have access to the facts and the general assertions used by the inference mechanism in its derivation.

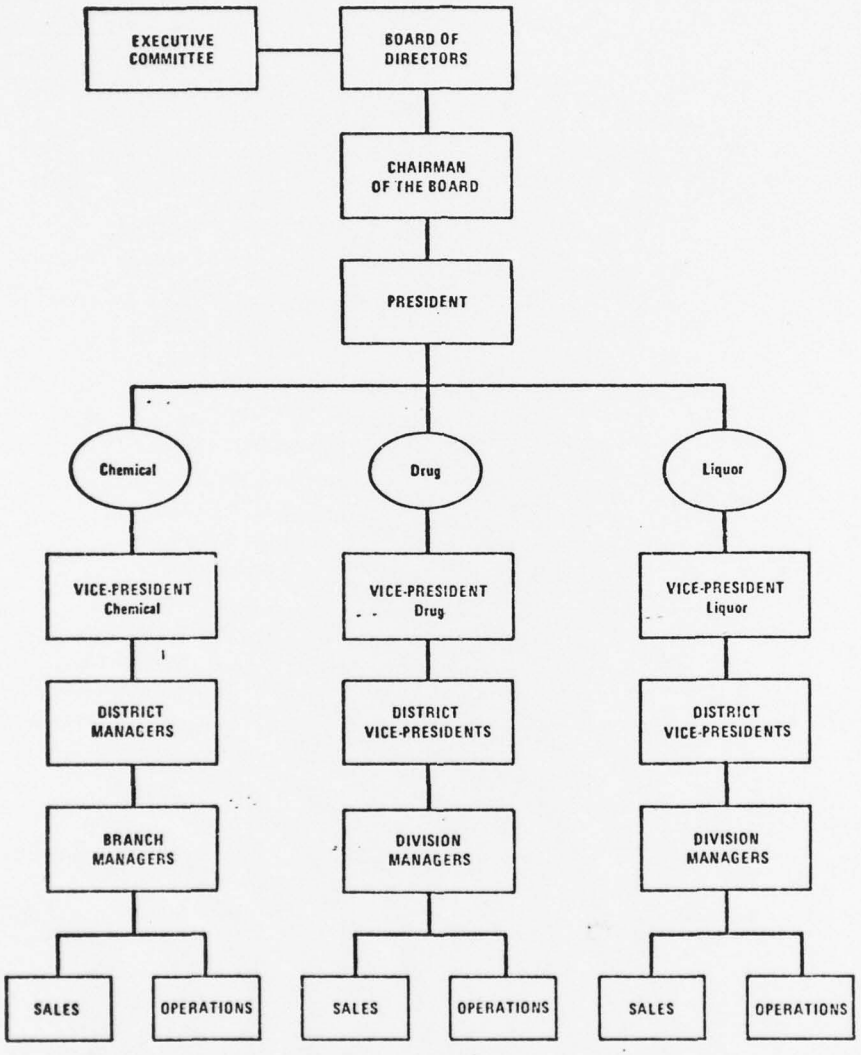

Figure 5. Organization Chart.

In Figures 6 and 7 we give an example set of specific facts and general assertions that are pertinent to questions of information flow. Of course, in an actual situation, there could be hundreds of additional general assertions in the system, and thousands of additional specific facts.

One successful derivation that our system could come up with, given the assertions and facts in Figures 6 and 7, is illustrated in Figure 8. When our system is given a complex question, it is broken down into a set of assumption predicates and a set of goal predicates. When both assumptions and goals exist, the Middle-Term Chain Generator is invoked to find middle-term predicate occurrences linking assumptions and goals. We will see an example of this process in the next example. For the current query we have only a single goal, namely, to establish information flow between Engler and Zembruski. In this situation the system will back up from the goal statement as illustrated in Figure 8 and select premises that can deductively lead to the establishment of the goal.

Each premise in Figure 8 represents an instance of a general assertion. The goal statement is enclosed in a rectangle, as are the two specific facts obtained from the data base. The vertical lines connecting the instances of predicates--for example, the line connecting two instances of Sibling--are unification arcs that are located by searching the predicate connection graph. It is within this graph that the system finds the linkages that enable it to link Brother to Sibling, Sibling to Relative, Relative to Nepotism, Nepotism to Friendship, and, finally, Friendship to Information-flow.

The general assertions used in Figure 8 are not all strictly true. For example, the premise concerning nepotism for relatives of subordinate employees is sometimes true (for certain circumscribed contexts or situations), but is clearly not always true. We anticipate many uses in our system of such plausible premises. Where this is done, it is clearly important to be able perspicuously to display to a user the lines of logical argument that are being followed by the system, so that the user can evaluate the credibility of a conclusion drawn from such questionable premises. Our system permits the discovery of alternative derivations for the same conclusion, which may enhance the credibility of the conclusion.

We note here a possible use of semantic advice. The user could suggest the use of particular premises or predicates that he feels may be appropriate to a particular query. For the current query, he may feel the premise concerning nepotism for relatives may be appropriate to establish information flow. The system would try

1. Zembruski is division head of Chemical: Head(Zembruski,Chemical)
2. Engler is division head of Liquor: Head(Engler,Liquor)
3. Richard Z. is a line subordinate of Engler: Line-sub(Richard Z.,Engler)
4. King is a line subordinate of Engler: Line-sub(King,Engler)
5. MR-Aces is a bridge club: Bridge-club(MR-Aces)
6. Rita S. is a member of MR-Aces: Member(Rita S.,MR-Aces)
7. Ann K. is a member of MR-Aces: Member(Ann K.,MR-Aces)
8. Rita S. is the wife of Smythe: Wife(Rita S.,Smythe)
9. Ann K. is the wife of King: Wife(Ann K.,King)
10. Richard Z. is the brother of Zembruski: Brother(Richard Z.,Zembruski)

Figure 6. Specific Facts.

1. Brothers and sisters are siblings.
   $\forall x,y$ (v(Brother(x,y), Sister(x,y)) $\Rightarrow$ Sibling(x,y))
2. Wives maintain information flow with their husbands.
   $\forall x,y$ (Wife(x,y) $\Rightarrow$ Info-flow(x,y))
3. Information flow runs between friends.
   $\forall x,y$ (Friend(x,y) $\Rightarrow$ Info-flow(x,y))
4. Every worker who is not a chairman of the board has a boss.
   $\forall x \exists y$ (&(Worker(x), $\neg$ Chairman(x)) $\Rightarrow$ Boss(y,x))
5. Every line subordinate is a worker and a subordinate.
   $\forall x,y$ (Line-sub(x,y) $\Rightarrow$ &(Worker(x), Subord(x,y)))
6. Every staff subordinate is a worker and a subordinate.
   $\forall x,y$ (Staff-sub(x,y) $\Rightarrow$ &(Worker(x), Subord(x,y)))
7. If someone does a nepotistic favor for another, then they are friends.
   $\forall x,y$ (Nepot(x,y) $\Rightarrow$ Friend(x,y))
8. Staff subordinates have information flow with their superiors.
   $\forall x,y$ (Staff-sub(x,y) $\Rightarrow$ Info-flow(x,y))
9. Line subordinates have information flow with their superiors.
   $\forall x,y$ (Line-sub(x,y) $\Rightarrow$ Info-flow(x,y))
10. There is a boss who has information flow with everyone of his subordinates.
    $\exists x \forall y$ (&(Boss(x,y), Subord(y,x)) $\Rightarrow$ Info-flow(x,y))
11. If someone is a subordinate of another, the superior may do a nepotistic favor for a relative of the subordinate.
    $\forall x,y,z$ (&(Subord(x,y), Relative(x,z)) $\Rightarrow$ Nepot(y,z))
12. People who are cousins or siblings are relatives.
    $\forall x,y$ (v(Cousin(x,y), Sibling(x,y)) $\Rightarrow$ Relative(x,y))
13. Members of a bridge club maintain information flow with each other.
    $\forall x,y,z$ (&(Bridge-club(x), Member(y,x), Member(z,x)) $\Rightarrow$ Info-flow(y,z))
14. The subordinate relation is transitive.
    $\forall x,y,z$ (&(Subord(x,y), Subord(y,z)) $\Rightarrow$ Subord(x,z))
15. Information flow is transitive.
    $\forall x,y,z$ (&(Info-flow(x,y), Info-flow(y,z)) $\Rightarrow$ Info-flow(x,z))
16. Information flow is symmetric.
    $\forall x,y$ (Info-flow(x,y) $\Rightarrow$ Info-flow(y,x))

Figure 7. General Assertions.

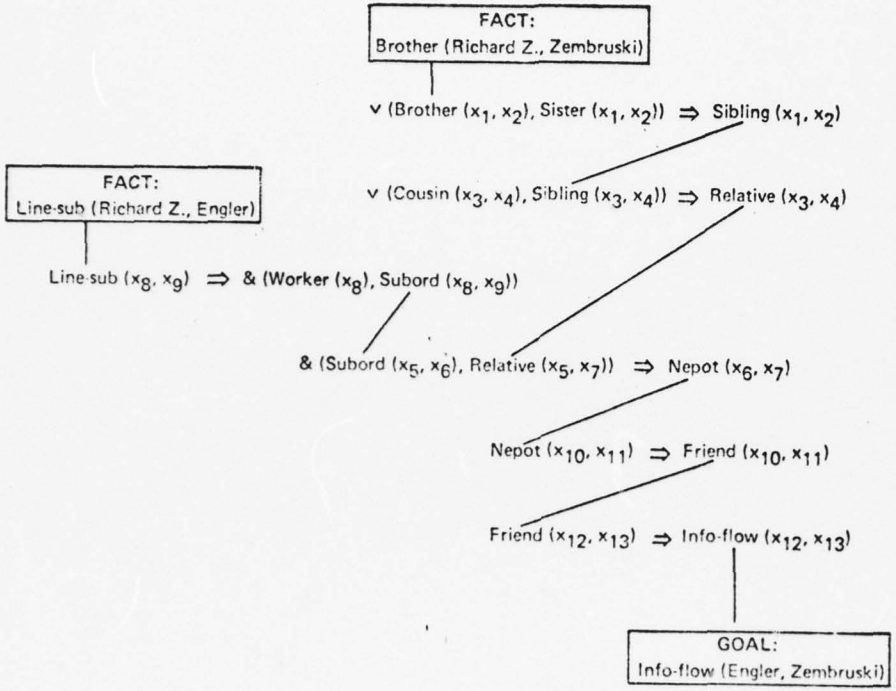

Figure 8. Derivation of Info-flow(Engler, Zembruski).

to use that premise in its proof. Similarly, if the user felt the relation "nepotism" is of key importance, the system would be alerted to link through occurrences of that predicate when possible. Note also that such advice could be placed in the advice file for general use (e.g., if goal is Info-flow, use Nepotism as middle term, or in more symbolic form "Goal(Info-flow); (Nepotism)").

To show another capability of the system, consider the following situation. We ask the system "What if Zembruski were to appoint Smythe to his staff; would there then be information flow between Engler and Zembruski?" Here we are suggesting the use of a particular assumption. Using this assumption the system will try to establish the same goal as in the earlier example.

A successful derivation for this query is shown in Figure 9. This proof uses premises quite different from those in the first derivation. This second derivation is more complex than the first. We can get the gist of the argument, however, if we follow through a few of the implications of the boxed specific facts that have been obtained from the data base. In order deductively to link Staff-sub(Smythe, Zembruski) to the goal Info-flow(Engler,Zembruski), the system has to determine that Smythe has a wife, Rita; that the MR-Aces is a bridge club; that Rita is a member of that bridge club; that Ann K. (the wife of employee King) belongs to the same bridge club, and hence, via a general assertion, has possible information flow with Rita; and that King is a line subordinate of Engler. Together, these relations deductively establish that there may indeed be information flow between Engler and Zembruski, namely through the wives of two of their subordinates.

We can use this example to show the basic operation of the system. The system is given an assumption and a goal. The Middle-Term Chain Generator attempts to find a chain of predicate occurrences that deductively link assumption to goal via the premises. The predicate connection graph, which contains information on the deductive connections between premises, is used in this chain-generation process. If the chain generator is successful and produces a chain, the Proof Proposal Generator extracts the set of premises containing the occurrences in the chain, and the system has the beginning of a proof plan. In Figure 9, the set of premises on the right were formed as a result of a chain linking the assumption to the goal via occurrences of the predicate Info-flow.

The Proof Proposal Generator then examines the set of premises to determine whether subproblems remain. In Figure 9, four subproblems were formed and are resolved using the four premises on the left. Subproblems resulting from these four premises are specified as needing fact-file support. (We have previously indicated to the system that certain predicates should be left for data-base search, either because we have complete knowledge about certain predicates such as Line-sub in an organization or because the information can be easily determined by the user, such as the wives of employees. In the latter case, the system would essentially be giving the user a "conditional answer," leaving certain subproblems open for user completion.)

Once all subproblems have been deductively resolved or left for fact-file support, the Proof Proposal Verifier combines the substitutions of all the unifications in the proof to check for consistency. Inconsistency occurs if a variable is required to take on two different constant values simultaneously. If verification is successful, the data management system is invoked to locate the specific facts needed for proof completion. In Figure 9, the six facts shown complete the proof.

While we have suggested the importance of displaying evidence for (or against) a deduced answer, we can readily see that derivations such as the one illustrated in Figure 9 may become difficult to follow. It is important to display machine-generated logical arguments in as perspicuous and user oriented a form as possible. We are currently developing techniques to display English-like formulations for such proofs.

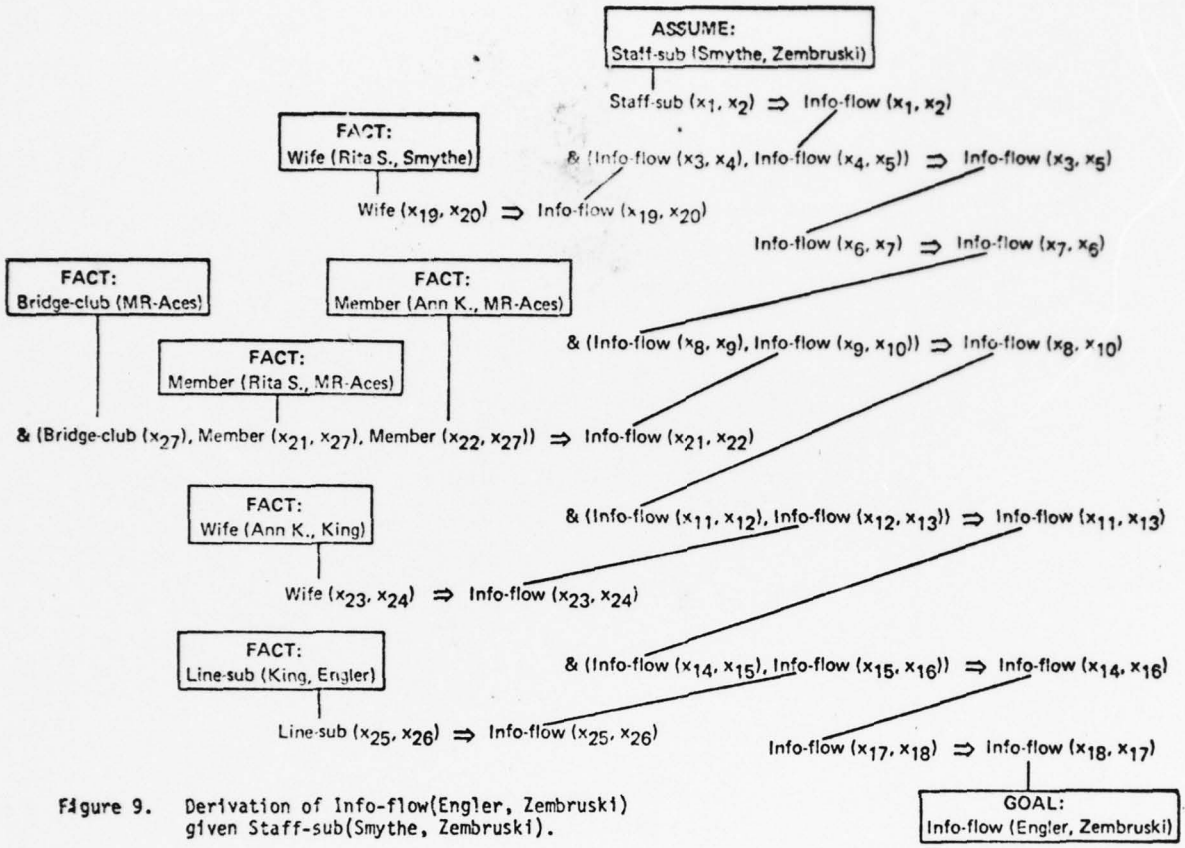

Figure 9. Derivation of Info-flow(Engler, Zembruski) given Staff-sub(Smythe, Zembruski).

SUMMARY

We have argued that inference mechanisms can significantly enhance the power and usability of a data management system. They enable the compact storage of a large amount of information in the form of general assertions, and they enable the combination of these assertions, with explicitly stored specific facts, to deduce other specific facts that would otherwise not be available.

Perhaps just as important, a deductive capability in a data management system can enable extendability of user language. The general assertions used by the deductive mechanisms can definitionally connect the concepts used by the data management system for organizing its data base to different concepts more appropriate for a particular user community.

We have briefly described a deductive system specifically designed to provide inferential capability for a data management system. From various files containing information extracted from general assertions, the system generates middle-term chains, which it combines into proof proposals. These proposals are then used in the generation of data-base search requests for concrete facts, which, in turn, transform proposals into complete proofs and answers.

Applying deduction to practical question-answering in realistic environments requires special attention to the previously unsolved problem of efficiently selecting, from very large files of specific facts and general assertions, the very few that are relevant for a particular deduction. Our approach to this selection problem involves constructing abstract proof plans and then iteratively fleshing them out with more and more detail. Particular facts and assertions are selected for trial only when they fit into some plan. Semantic advice, i.e., advice specific to a particular subject domain, can be used to guide the construction and articulation of proof plans. On the basis of our experiments so far, the approach looks promising.

Acknowledgments: This work has been supported in part by the Advanced Research Projects Agency and System Development Corporation.

REFERENCES

Bledsoe, W.W., and Bruell, P. (1974). A Man-Machine Theorem-Proving System. Artificial Intelligence, 5, No. 1, 51-72.

Chang, C.L., and Lee, R.C.T. (1973). Symbolic Logic and Mechanical Theorem Proving. Academic Press, New York.

Codd, E.F. (1970). A Relational Model of Data for Large Shared Data Banks. Comm. ACM, 13, No. 6, 377-387.

Date, C.J. (1975). An Introduction to Database Systems. Addison-Wesley, Reading, Mass.

Elliott, R.W. (1965). A Model for a Fact Retrieval System. TNN-42, Computation Center, U. of Texas, Austin.

Fahlman, S. (1975). A System for Representing and Using Real-World Knowledge. MIT AI Memo 331, Cambridge, Mass.

Green, C.C. (1969). Theorem Proving by Resolution as a Basis for Question-Answering Systems. In Machine Intelligence 4, Meltzer and Michie (Eds.), Edinburgh U. Press, Edinburgh, 183-205.

Hewitt, C. (1971). Procedural Embedding of Knowledge in PLANNER. Proc. Second Intl. Jt. Conf. Artificial Intelligence, British Computer Society, London, 167-184.

Kellogg, C.H. (1968). A Natural Language Compiler for On-line Data Management. Proc. FJCC, 33, Thompson Book Co., Washington, 473-493.

Kellogg, C.H., Burger, J., Diller, T., and Fogt, K. (1971). The CONVERSE Natural Language Data Management System: Current Status and Plans. Proc. Symp. Information Storage and Retrieval, U. of Maryland, College Park, 33-46.

Klahr, P. (1975). The Deductive Pathfinder: Creating Derivation Plans for Inferential Question-Answering. PH.D. Dissertation, Computer Sciences Dept., U. of Wisconsin, Madison.

Kowalski, R. (1975). A Proof Procedure Using Connection Graphs. JACM, 22, No. 4, 572-595.

McDermott, D.V. (1975). Very Large PLANNER-type Data Bases. MIT AI Memo 339, Cambridge.

Minker, J., Fishman, D.H., and McSkimin, J.R. (1973). The Q* Algorithm--A Search Strategy for a Deductive Question-Answering System. Artificial Intelligence, 4, No. 3/4, 225-243.

Nevins, A.J. (1974). A Human Oriented Logic for Automatic Theorem Proving. JACM, 21, No. 4, 606-621.

Raphael, B. (1964). A Computer Program Which "Understands." Proc. FJCC, 26, Spartan Press, Baltimore, 577-589.

Robinson, J.A. (1965). A Machine-Oriented Logic Based on the Resolution Principle. JACM, 12, No. 1, 23-41.

Schwarcz, R.M., Burger, J.F., and Simmons, R.F. (1970). A Deductive Question-Answerer for Natural Language Inference. Comm. ACM, 13, No. 3, 167-183.

Shostak, R.F. (1976). Refutation Graphs. Artificial Intelligence, 7, No. 1, 51-64.

Travis, L., Kellogg, C., and Klahr, P. (1973). Inferential Question-Answering: Extending CONVERSE. SP-3679, System Development Corporation, Santa Monica.

Yates, R.A., Raphael, B., and Hart, T.P. (1970). Resolution Graphs. Artificial Intelligence, 1, No. 4, 257-289.